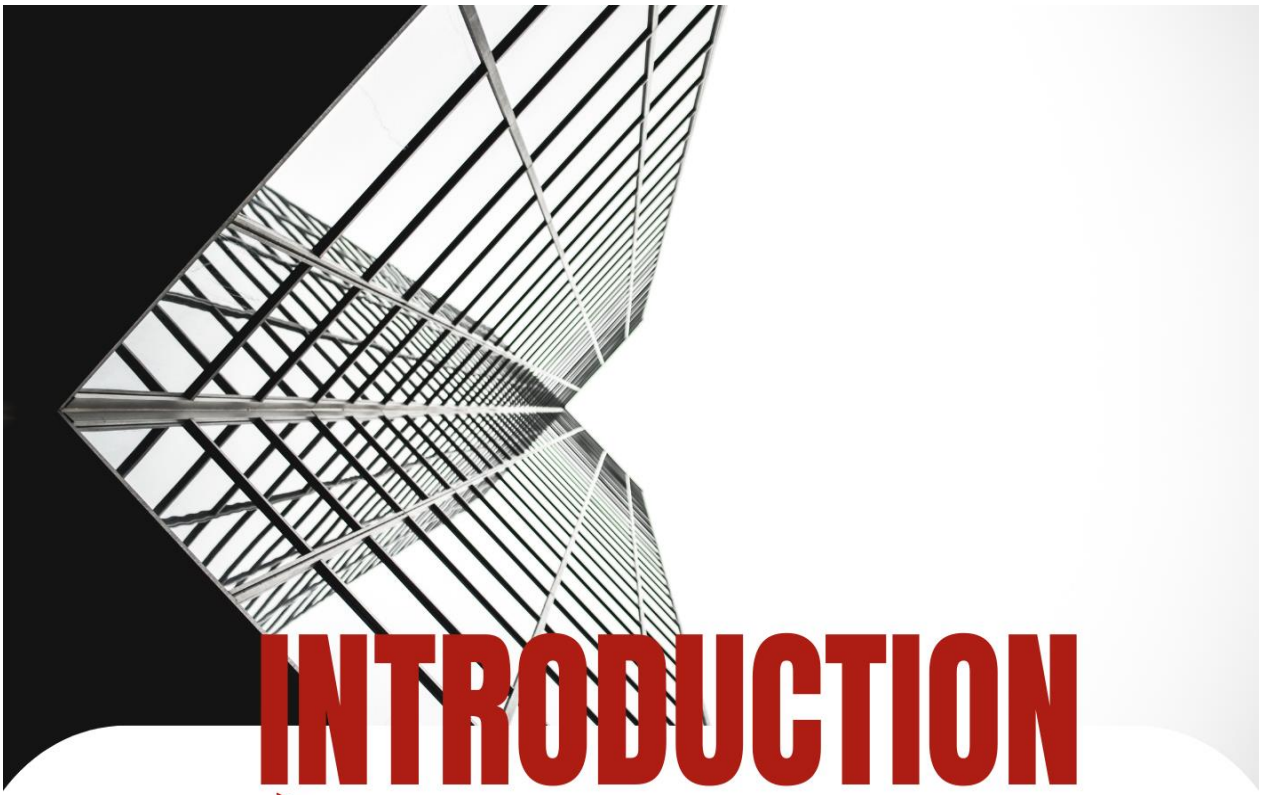

INTRODUCTION À JAVASCRIPT

Un guide pour apprendre les bases du langage

SIMPLON
ACADEMY

Houda LOQMANE

Novembre 2024

TABLE DE MATIERES

DEFINITION DU JAVASCRIPT

JavaScript est un langage de programmation utilisé pour rendre les pages web interactives, comme gérer les événements, animer des éléments ou mettre à jour le contenu en temps réel.

IMPORTANCE DE JAVASCRIPT DANS LE DEVELOPPEMENT WEB

1.Frontend : Dynamise les pages web et facilite le développement d'interfaces interactives avec des frameworks comme React ou Vue.js.

2.Backend : Grâce à Node.js, il permet de créer des serveurs performants et des applications en temps réel, tout en utilisant le même langage pour le client et le serveur.

DECLARATION DES VARIABLES

En JavaScript, on peut déclarer des variables à l'aide des mots-clés :

```
var name = "Houda";

let age = 22;

const pays = "Maroc";
```

La différence entre eux est :

- **let** : pour déclarer des variables dont la valeur est modifiable, avec une portée limitée au bloc.
- **const** : pour des valeurs constantes ou des objets qui ne doivent pas être réassignés.
- **var** : à éviter, sauf pour maintenir du code ancien, car il a une portée globale.

TYPES DE DONNEES

Il existe plusieurs types de données primitifs et références dans lesquels les variables peuvent être stockées.

TYPES PRIMITIFS :

- String : Représente des chaînes de caractères.
- Number : Représente des valeurs numériques, entières ou à virgule flottante.
- Boolean : Représente une valeur logique, soit true (vrai) soit false (faux).
- Undefined : Une variable qui a été déclarée mais n'a pas encore été assignée à une valeur.
- Null : Représente l'absence délibérée de valeur ou un objet vide.

```
let name = "Houda";//String
let age = 22;//Number
let estEtudiante = true;//Boolean

let valeur;
console.log(valeur);// Affiche Undefined

let objet = null; // Null
```

TYPES REFERENCES :

- Array : Un type d'objet spécifique qui permet de stocker des listes ordonnées.

```
let fruits = ["framboise", "mangue", "avocat"]
console.log(fruits[0]); // Affiche framboise
```

- Object : Un type complexe qui peut contenir plusieurs valeurs sous forme de paires clé-valeur (propriétés).

```
const utilisateur= {
    name:"Houda",
    age: 22,
    profession : "Développeuse"
};

console.log(utilisateur.age);// Affiche 22
console.log(utilisateur["profession"]);// Affiche Développeuse
```

On peut accéder aux propriétés d'un objet de deux manières : par la notation par point (.) ou par la notation par crochets ([]).

LES CONDITIONS

Les conditions en JavaScript permettent d'exécuter des blocs de code différents selon que certaines expressions soient vraies ou fausses.

Les principales conditions sont les suivantes :

- if : Vérifie une condition et exécute un bloc de code si elle est vraie.
- else : Exécute un bloc de code si la condition de l'if est fausse.
- else if : Permet de tester d'autres conditions après le premier if.
- switch : Permet de tester une expression contre plusieurs valeurs possibles.

EXEMPLE DE CONDITION AVEC UN « IF » :

➢ Permet d'exécuter un bloc de code si une condition est vraie.

```
let age = 22
if(age>=18){
    console.log("vous êtes majeur");// Affiche vous êtes majeur
}
```

EXEMPLE DE CONDITION AVEC UN « IF…ELSE »

➢ Permet d'exécuter un bloc de code si la condition dans if est fausse.

```
let age = 22
if(age>=18){
    console.log("vous êtes majeur");//si age est supérieur a 18, il affiche vous êtes majeur
}else{
    console.log("vous êtes mineur");//si age est inferieur a 18 ,il affiche vous êtes mineur
}
```

EXEMPLE DE CONDITION AVEC UN « **ELSE…IF** »

- Permet de tester plusieurs conditions, en vérifiant les conditions les unes après les autres.

```
let age = 100
if (age < 18) {
    console.log("Vous êtes mineur.");
} else if (age >= 18 && age < 65) {
    console.log("Vous êtes adulte.");
} else {
    console.log("Vous êtes senior.");
}
```

EXEMPLE DE CONDITION AVEC UN « **SWITCH** »

- Permet de tester plusieurs valeurs possibles d'une variable ou expression et d'exécuter un bloc de code correspondant à la valeur trouvée.

```
let day = 7
switch(day){
    case 1:
        console.log("Monday");
        break;
    case 2:
        console.log("Tuesday");
        break;
    case 3:
        console.log("Wensday");
        break;
    case 4:
        console.log("Thursday");
        break;
    case 5:
        console.log("Friday");
        break;
    case 6:
        console.log("Saturday");
        break;
    default:
        console.log("Sunday");
}
```

LES BOUCLES

Les boucles permettent de répéter un bloc de code automatiquement, selon une condition ou un nombre d'itérations défini.

En JavaScript, il existe 4 types principaux de boucles :

- For : Utilisée pour répéter un bloc de code un nombre déterminé de fois.

```
for (let x = 1; x <= 10; x++){
    console.log(x);
}
```

- While : Répète un bloc de code tant qu'une condition est vraie.

```
let compteur = 0
while(compteur<5){
    console.log(compteur);
    compteur++
}
```

- do...while : Exécute un bloc de code au moins une fois, puis répète tant qu'une condition est vraie.

```
let compteur = 0
do{
console.log(compteur);
compteur++
}while(compteur<5)
```

- for...of et for...in :

for...of : Parcourt les valeurs d'objets itérables (tableaux, chaînes, etc.).

```
let fruits = ["framboise", "pomme","fraise"];
for(let fruit of fruits){
    console.log(fruit);

}
```

for...in : Parcourt les clés (ou propriétés) d'un objet.

```
const utilisateur = {
    nom: "Alice",
    âge: 25,
    profession: "Développeuse"
};

for (const clé in utilisateur) {
    console.log(`${clé} : ${utilisateur[clé]}`);
}
```

LES OPERATEURS

En JavaScript, il existe plusieurs catégories d'opérateurs, chacun servant à effectuer des opérations spécifiques. Voici les principales catégories et leurs types :

OPERATEURS ARITHMETIQUES

Utilisés pour effectuer des calculs mathématiques.

- **Addition** : **+**
- **Soustraction** : **-**
- **Multiplication** : *
- **Division** : /
- **Modulo** (reste de la division) : **%**
- **Exponentiation** : **
- **Incrémentation** : ++

- **Décrémentation** : **--**

OPERATEURS DE COMPARAISON

Utilisés pour comparer deux valeurs, souvent dans des conditions.

- **Égal à** : **==**
- **Strictement égal à** : **===**
- **Différent de** : **!=**
- **Strictement différent de** : **!==**
- **Plus grand que** : **>**
- **Plus petit que** : **<**
- **Plus grand ou égal à** : **>=**
- **Plus petit ou égal à** : **<=**

OPERATEURS LOGIQUES

Utilisés pour combiner ou inverser des conditions.

- **ET logique** : &&
- **OU logique** : ||
- **NON logique** : !

OPERATEURS TERNAIRES

Permettent une condition en une seule ligne :

- `condition ? valeurSiVrai : valeurSiFaux`

```
let age = 20;
let message = (age >= 18) ? "Vous êtes majeur." : "vous êtes mineur.";
console.log(message);
```

OPERATEURS DE TYPE

Typeof : Retourne le type d'une variable : typeof variable

```
let y ="hello world"
console.log(typeof y);// String
```

LES FONCTIONS

En JavaScript, une fonction est un bloc de code réutilisable qui exécute une tâche spécifique. Elle peut recevoir des paramètres en entrée et peut retourner une valeur.

FONCTION DECLAREE

- Une fonction définie avec le mot-clé function :

```
function addition(a, b) {
    return a + b;
}
console.log(addition(5, 3)); // Affiche : 8
```

FONCTION FLECHEE (ARROW FUNCTION)

- Syntaxe simplifiée pour écrire des fonctions

```
const addition = (a, b) => a + b;
console.log(addition(5, 3)); // Affiche : 8
```

LES FONCTIONS PREDEFINIES

En JavaScript, il existe de nombreuses fonctions prédéfinies qui permettent de manipuler des chaînes de caractères, des nombres, des tableaux, des objets, et d'autres éléments. Voici un aperçu des principales fonctions classées par catégories :

MANIPULATION DE CHAINES DE CARACTERES (STRING) :

- toUpperCase() : Convertit une chaîne en majuscules.

```
let text = "hello";
console.log(text.toUpperCase());//HELLO
```

- toLowerCase() : Convertit une chaîne en minuscules.

```
let text = "HELLO";
console.log(text.toLowerCase());//hello
```

- split(separator) : Divise une chaîne en un tableau selon un séparateur.

```
let text = "a,b,c";
console.log(text.split(","));//[ 'a', 'b', 'c' ]
```

- charAt(index) : Retourne le caractère à une position donnée.

```
let text = "Hello";
console.log(text.charAt(1));// "e"
```

MANIPULATION DES TABLEAUX (ARRAY) :

- push(element) : Ajoute un élément à la fin d'un tableau (.push())
- pop() : Supprime le dernier élément d'un tableau (.pop())
- shift() : Supprime le premier élément d'un tableau (.shift())
- unshift(element) : Ajoute un élément au début du tableau (.unshift())
- slice(start, end) : Extrait une portion du tableau.

```
let fruits = ["Pomme", "Banane", "Orange"];

fruits.push("Mangue");
console.log(fruits);
// Résultat : ["Pomme", "Banane", "Orange", "Mangue"]


let removedFruit = fruits.pop();
console.log(fruits);
// Résultat : ["Pomme", "Banane", "Orange"]
// Élément supprimé : "Mangue"


fruits.unshift("Fraise");
console.log(fruits);
// Résultat : ["Fraise", "Pomme", "Banane", "Orange"]


removedFruit = fruits.shift();
console.log(fruits);
// Résultat : ["Pomme", "Banane", "Orange"]
// Élément supprimé : "Fraise"


let slicedFruits = fruits.slice(1, 3); // Extraire les éléments aux indices 1 et 2
console.log(slicedFruits);
// Résultat : ["Banane", "Orange"]
```

- .map() : Elle est utilisée pour créer un nouveau tableau en appliquant une fonction à chaque élément du tableau d'origine.

```
let numbers = [1, 2, 3, 4];
let multiplication = numbers.map(num => num * num);
console.log(multiplication);
// Résultat : [1, 4, 9, 16]
```

- .reduce() : Elle permet de réduire un tableau à une seule valeur en appliquant une fonction de cumul.

```
let numbers = [1, 2, 3, 4];
let sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum);
// Résultat : 10
```

MANIPULATION DES OBJETS(OBJECT) :

- Object.keys() :Cette méthode retourne un tableau contenant toutes les clés (propriétés) d'un objet.

```
let person = {
    name: "Alice",
    age: 25,
    city: "Paris",
  };

  console.log(Object.keys(person));
  // Résultat : ["name", "age", "city"]
```

- Object.values() :Cette méthode retourne un tableau contenant toutes les valeurs des propriétés d'un objet.

```
let person = {
    name: "Alice",
    age: 25,
    city: "Paris",
  };

  console.log(Object.values(person));
  // Résultat : ["Alice", 25, "Paris"]
```

GIT / GITHUB

- Git : Un système de contrôle de version distribué permettant de suivre les modifications du code.
- GitHub : Une plateforme basée sur Git pour héberger et collaborer sur des projets de programmation.

CONCEPTS DE BASE :

➢ Git :

- Repository (Dépôt) : Un espace où le code source est stocké.
- Commit : Une sauvegarde d'une version du code.
- Branch (Branche) : Une ligne indépendante de développement.
- Merge : Fusion de branches.

➢ GitHub :

- Dépôt distant.
- Collaboration à travers les pull requests.
- Gestion des problèmes.

COMMANDES GIT ESSENTIELLES :

1. Git init
2. Git add .
3. Git commit -m « Message »
4. Git push

UTILISATION DU GITHUB :

➢ Créer un dépôt sur GitHub :

- Connecte-toi à GitHub.
- Crée un nouveau dépôt en ligne.

- Connecter un dépôt local à GitHub.
- Collaborer via les Pull Requests.

CONCLUSION

Dans cette documentation, nous avons exploré les bases de JavaScript, y compris les types de données, les structures de contrôle, les fonctions, et les manipulations. Ces concepts constituent les fondations essentielles pour développer des applications interactives.

Merci d'avoir lu cette documentation. Si vous avez des suggestions ou des corrections, n'hésitez pas à contribuer via GitHub ou à me contacter directement.